Scripting for HTML

Objectives

- Understand scripting for HTML
- Create a script
- Debug a script
- Understand objects
- Use JavaScript event handlers
- Create a function
- Assign a variable
- Create a conditional

HTML allows you to create basic Web pages easily. Using many of the newest features available in Web page design, however, requires the incorporation of **scripts**, which are programs in a Web page that run on the viewer's browser. Using scripts can expand the number of features you can add to your Web pages and can simplify some HTML commands. Lydia Burgos works in the information systems division of Nomad Ltd. Her supervisor heard about the advantages of scripts at a recent seminar and has asked Lydia to create a report on ways that scripts could enhance the company's current Web publication. Lydia begins by researching the fundamentals of **scripting**, which is the process of writing scripts.

Understanding Scripting for HTML

A **script** is a small program contained within an HTML document that can be executed by a Web browser. In the early days of the Web, scripts were separate files stored and run on a remote Web server. Today, Web page designers can place scripts within a page's standard HTML, and the scripts can run on each user's computer. This change is made possible by new techniques for data transfer on the Web as well as a new generation of more powerful home computers. Other forms of Web page programming (such as CGI files that process data from Web page forms) may also be referred to as scripts. However, scripts that run on a user's browser represent the most powerful and innovative form of scripting on the Web today. This is the definition we will use in this book. Figure H-1 shows the source code for a Web page containing a script, as well as the Web page generated by that code. In researching scripting fundamentals, Lydia Burgos has learned that using scripting in Web page development offers several advantages over using HTML alone. These advantages include the following:

Flexibility

Scripting allows you to modify your Web pages in ways that are not yet part of standard HTML. Additionally, you can combine scripting codes in many different ways, which opens the door to more variety and creativity in your pages than with HTML alone. For example, you can have a message that runs automatically when the Web page is opened and that is updated automatically, such as a current list of top headlines for a news Web site.

Simplification

You can script some tasks that you usually would code using HTML. In many cases, scripting is more efficient and allows you to create a more organized publication. One of the first areas in which scripting is replacing HTML is in assigning styles to text. Instead of adding attributes to the tags for each block of text in your publication, you can insert a small script that automatically assigns specific attributes, such as boldface type or text color, to specific blocks of text throughout an entire Web page or set of pages.

Immediate response

Traditionally, to access many features available on a Web page, your browser contacts the Web site where the page resides and requests a linked Web page or new information based on your input. Embedded scripts eliminate the lag time involved in contacting the remote server, waiting for a response, and downloading new information. This is because the scripts run on your local computer and can change what your browser displays without downloading new information from the server.

Improved interactivity

The quick response time that local scripting provides allows you to incorporate an impressive amount of user interactivity into your Web pages. Since script processing occurs locally, your script can react almost instantaneously to any user action. For example, your script could display a short summary of a link when the user simply moves the pointer over the link without clicking it.

Reduced server load

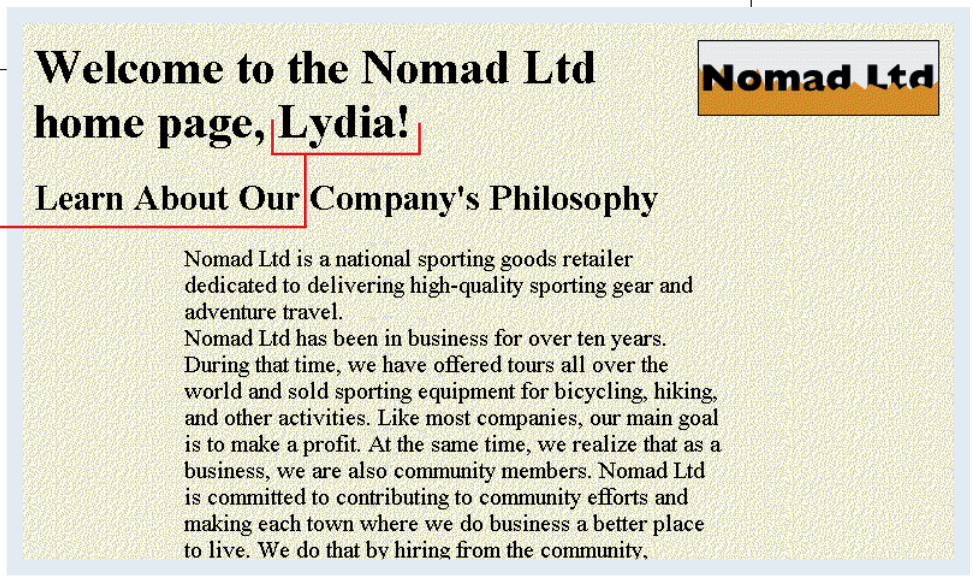
Local execution of scripts is an advantage not just for users, but for Web site administrators as well. The reduced demand on the server when some of the processing is shifted to local computers results in more free system resources. This may result in a decrease in download time for people viewing your site's pages and faster processing when a Web page does need to submit a request to the server.

FIGURE H-1: Web page script source code and browser document

```
<HTML>

<HEAD>
<TITLE>Nomad Ltd online</TITLE>

<SCRIPT LANGUAGE="javascript">
<!--
var name=prompt("For personal service, please type your first name and click
OK.","")

function submitted() {
      alert("Information submitted!")
}

function clearUp() {
      document.info.elements[0].value=""
      document.info.elements[1].value=""
      document.info.elements[2].value=""
      document.info.elements[3].value=""
      document.info.elements[4].value=""
      document.info.elements[5].value=""
}

//-->
</SCRIPT>
```

Script embedded in HTML code

User name generated by script



Choosing a scripting language

Several languages are available when writing scripts for an HTML document. These include **JavaScript** and **JScript**, which are both adaptations of Sun Microsystems' Java programming language, as well as **VBScript**, which is Microsoft's adaptation of its Visual Basic programming language for Web use. Both Internet Explorer 4 and Netscape Navigator 4 are largely compatible with JavaScript. On the other hand, JScript and VBScript are not universally compatible, meaning that users of certain browsers might be unable to view the scripted elements of your page if you used one of these languages. This text uses JavaScript to create pages viewable by both Internet Explorer and Netscape Navigator users.

Unit H
HTML

Creating a Script

You can add a script to a Web page just as you would add to or edit the Web page's HTML code. To do this, you can use a text editor such as WordPad or Notepad or a Web page editor such as FrontPage Express or Composer. In order for a browser to recognize a script, the script must be contained within <SCRIPT> HTML tags. It is also good practice to surround each script with a set of HTML tags to make it invisible to browsers incompatible with scripts. This second set of HTML tags tells incompatible browsers to bypass the script. While researching scripting fundamentals, Lydia found a sample script that validates a form before submitting it to a server by checking the form to be sure information has been entered in each field. If information has not been entered in each field and the user tries to send the form, then this form validation script displays a warning box which reminds the user to complete each field. Lydia decides to try adding this script to the Nomad Ltd home page in order to better understand how to insert scripts into HTML code.

Steps

QuickTip

When saving a file as HTML, be sure to select Text (.txt) as the file type to be saved and include the extension .htm in the filename.

1. Start your text editor program, open the file **HTML H-1.htm**, then save it as a text document with the filename **Scripted page.htm**
   If you use WordPad or Notepad, which are text editors built into Windows 95, the source code for the Web page appears as text in the text editor. If you use an HTML editing tool, such as Microsoft FrontPage or Netscape Composer, you will see the graphical representation of the file code. You must select the option Page Source or HTML from the View menu to see the HTML code for the page.
2. Select the text **[replace with opening script tags]** below the TITLE tag, then press **[Delete]**
3. Type **<SCRIPT LANGUAGE="javascript">** and press **[Enter]**
   Now the beginning of the script is marked with the <SCRIPT> HTML tag and an attribute specifying the scripting language you're using, which is JavaScript.

QuickTip

All browsers ignore the remainder of the line after the <!-- tag; thus, some script writers add a normal language comment here, such as "HIDE," to make the script code easier for programmers to look at and understand.

4. Type **<!--** and press **[Enter]** twice
   The tag <!-- tells an incompatible browser (one that can't process your script) to ignore the code that follows. A compatible browser (one that can interpret your script) will go ahead and process the script that follows.
5. Select the text **[replace with closing script tags]**, then press **[Delete]**
6. Type **//-->** and press **[Enter]**, then type **</SCRIPT>** and press **[Enter]**
   These tags mark the end of the script. Figure H-2 shows the Web page source code containing the opening and closing script tags and the dialog box generated by the script.
7. Check the lines you added for errors, make changes as necessary, then save Scripted page.htm as a text document

Trouble?

If the Web page does not display correctly, or if a dialog box opens describing an error in your script, click OK if necessary, read the next lesson to learn about debugging scripts, then return to look for errors in your Web page code.

8. Start your Web browser program, cancel any dial-up activities, open the page **Scripted page.htm**, scroll down to the text fields near the bottom of the page, fill in sample information, then click the **Send now! button**
   The script runs and displays the dialog box shown in Figure H-2.
9. Click the **OK button**
   The browser simulates form submission to a server. This script will save time for people using the Nomad Ltd Web page, since they do not have to wait for the form to be submitted to the server and then wait for an error message to be transmitted back to their computer.

FIGURE H-2: Source code for Web page dialog box

```
<HTML>

<HEAD>
<TITLE>Nomad Ltd online</TITLE>

<SCRIPT LANGUAGE="javascript">
<!--
function submitted() {
      alert("Information submitted!")
}
//-->
</SCRIPT>

</HEAD>

<BODY BACKGROUND="egg shell.jpg">

<IMG SRC="nomad.jpg" ALIGN="right" WIDTH=201 HEIGHT=63>

<H1>Nomad Ltd Home Page!</H1>

<H2>Learn About Our Company's Philosophy</H2>

<DIV ID=text>
Nomad Ltd has b
time, we have c
```

Opening script tags

Closing script tags

Script-generated dialog box

Types Of Nomad Tours

- Athlete: For those who are sports participants.
- Arts: Dedicated to enjoying and appreciating the performing arts.
- Leisure: Designed to let you relax and rejuvenate.

Tell Us More About Y

Microsoft Internet Explorer

Information submitted!

OK

First Name: Lydia

Street Address: 733 Baker Street

City: Springfield State:

Click here to submit this information. Send now!

Contact Us Directly!

Find out more about what we have to offer. Contact the tour office at Tourdirector@nomadltd.com

Linking to an external script

In addition to typing script code directly into a Web page, you can add scripts to your pages by using an HTML code that references a separate file containing script code. A script located in an external file that you can link to a Web page is known as a **scriptlet**. Scriptlets make it easier to share code and to reuse scripts on multiple pages by allowing you to use a script without needing to paste its code into each Web page.

Debugging a Script

No matter how carefully scripting code is entered, a script often contains errors, or **bugs**, the first time your browser processes it. In general, a bug causes the script to return unexpected and undesired results. These may include improper text formatting, the display of HTML or JavaScript code in the browser window, or, in the worst case, requiring you to exit and restart the browser. It's doubtful that simple errors you make entering the scripts in this book could hang your browser. However, when your results are different than those shown in this text, you can use a process called **debugging** to systematically identify and fix your script's bugs. Lydia reads up on debugging scripts and learns about several main culprits that cause undesired results. These are illustrated in Figure H-3.

Details

QuickTip

You can type HTML tags and property names in either capital or lowercase without affecting the browser's interpretation. Only scripting languages—not HTML—are case-sensitive.

Capitalization

JavaScript is a **case-sensitive** language, meaning that it treats capital and lowercase versions of the same letter as different characters. Thus, depending on the script being entered, capitalizing or not capitalizing letters can result in errors. Always check that you have used capital letters where called for and that all the capital letters in your script are correct.

Spacing

When entering script code from a printed source, it is easy to add extra spaces or to leave spaces out. Some parts of JavaScript syntax allow for extra spaces, which makes code easier to view and understand. However, as in HTML, incorrect spacing in certain parts of a script can render otherwise-perfect code incomprehensible to the browser. The best way to avoid spacing errors is to pay careful attention to spacing while you enter code.

Parentheses ( ), brackets [ ], braces { }, and quotes " "

JavaScript often uses these four types of symbols to enclose arguments, values, or numbers upon which certain commands are executed. In more complex scripts, you may end up with several sets enclosing other sets, which makes it easy to forget to type one or more closing symbols. In lines or blocks of script where these symbols are concentrated, it can be difficult to check for accuracy. However, to ensure that your script will run properly, you must make sure that each one is paired with its counterpart.

Typographical errors (0 vs. O, 1 vs. I)

Another common source of errors when entering code from printed material is interchanging characters that appear similar, such as the number 0 and the letter O. Generally, the context provides good clues to figuring this out. For example, even if you don't know the function of the expression "value=0", the use of the word *value* gives you a good clue to type a number rather than a letter. Ultimately, the best way to avoid confusing these symbols is to analyze what the script is doing in the current line. Once you understand this, it should be obvious which character to type.

Others

Inevitably, various other sources of error will creep into your scripts. To help you deal with miscellaneous problems, as well as those detailed earlier, JavaScript-compatible browsers often display a JavaScript Error window, as shown in Figure H-4, when they can't interpret your code. Generally, the window describes the type of error the browser encountered and the error's location, referenced by its line number in the script. With this information, some investigation of your code, and a bit of thought, you can track down and fix most any bug that crops up.

Now that you understand the basics of working with scripts, you can begin to learn how to construct your own scripts.

FIGURE H-3: JavaScript containing common bugs

```
<HTML>

<HEAD>
<TITLE>Nomad Ltd online</TITLE>
<SCRIPT LANGUAGE="javascript">
<!--
function submit() {}

function verify() {
      if (document.info.elements[O].value == "" ||
      document.info.elements[1].value == "" ||
      Document.info.elements[2].value == "" ||
      document.info.elements[3].value == "" ||
      document. info.elements[4].value == "" ||
      document.info.elements[5].value == "" {
            alert("Please complete each field")
      }     else {
            submit()
      }
}
//-->
</SCRIPT>

</HEAD>

<BODY BACKGROUND="egg shell.jpg">
```

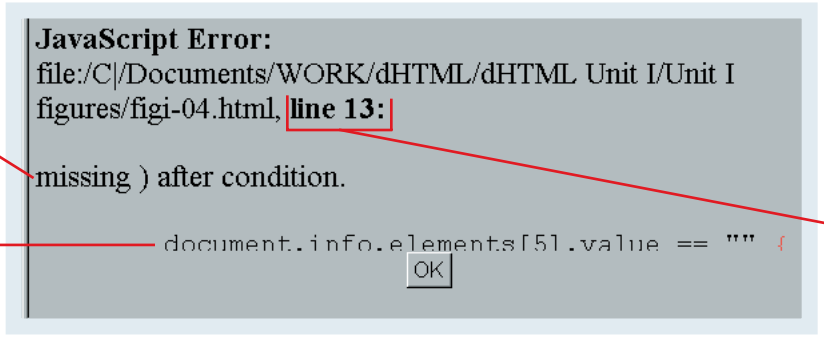
Unclosed parentheses

Incorrect capital

Extra space

Letter instead of number

FIGURE H-4: JavaScript Error window



Description of error

Erroneous code with error indicated

Location of error in code

Commenting a script

While writing a script, it can be useful to include comments in ordinary language to explain what the script is doing at particular places. These notes can be helpful for you when editing or debugging your script or for someone else who wants to understand how the script works. To add a comment to the end of a line of the script, type // at the end of the line's script code, then type your comment. You also can mark several lines of text in a script as a comment by typing /* at the start of the block and */ at the end.

Understanding Objects

In order to organize and work with the various parts of the browser window, including the current Web page and each of its elements, JavaScript treats each element in the window as a unit called an **object**. JavaScript gives each object a default name and set of descriptive features based on its location and function. These features include **properties**, which are qualities such as size, location, and type. In a Web page containing a graphic, the IMG tag would be an object, and the WIDTH setting would be one of its properties. Each object also has associated **methods**, which are actions the object can carry out. For example, each frame in a browser window has an associated ALERT method, which allows you to create customized dialog boxes. JavaScript organizes objects in an **object hierarchy**, much like the system of folders used by Windows to keep track of disk contents. As Lydia researches the object hierarchy, she learns different ways to refer to the various elements in a document when writing a script.

To specify an object on a Web page, you need to detail its position in the hierarchy, beginning on the document level and then separating each level name with a period. This method of referencing objects in the hierarchy is called **dot syntax**. For example, to specify an image on a Web page, the name would begin *document.images*. Because a document can contain more than one image, each image is assigned a number, based on the order in which it appears in the Web page code. Other elements such as anchors, hyperlinks, and forms also are assigned numbers. It is important to note, however, that in JavaScript, this numbering begins with 0. Thus, when writing a script that refers to the first image in your HTML code, you would use the name *document.images[0]*. The second image in your HTML code would be called *document.images[1]* according to the object hierarchy. To specify an element such as a button or a text field contained inside a form you would use a slightly longer address. For example *document.forms[0].elements[0]* would refer to the first element created in the first form in a Web document.

You also use dot syntax to refer to an object's methods. For example, the code *document.write("Copyright Course Technology, 1999.")* calls on the *write* method of the *document* object. This code causes the information in parenthesis to be written to the document. Table H-1 lists the default objects that are part of every browser window, along with their methods.

TABLE H-1: Default JavaScript objects and methods

object name	method name	description
window	alert(message)	Displays a dialog box with a message in the window
	close()	Closes the window
	prompt(message, default_text)	Displays a dialog box prompting the user for information
	scroll(x, y)	Scrolls to the x.y coordinate in the window
frame	alert(message)	Displays a dialog box with a message in the frame
	close()	Closes the frame
	prompt(message, default_text)	Displays a dialog box prompting the user for information
history	back()	Returns to the previous page in the history list
	forward()	Goes to the next page in the history list
location	reload()	Reloads the current page
document	write()	Writes text and HTML tags to the current document
	writeln()	Writes text and HTML tags to the current document on a new line
form	reset()	Resets the form
	submit()	Submits the form

Naming an object

When you initially create a page element, you can assign it an ID using the ID property. Then, you can refer to the object by its name instead of using its index number. For example, the code <*FORM ID="input"*> assigns the ID *input* to the form. You can then use the form ID *input* rather than *forms[0]*to refer to the form or any elements within it. You could now refer to this form as *document.input.elements[0]* rather than *document.forms[0].elements[0]*. By assigning a name to each element, such as check boxes or text boxes, you could eliminate the need for index numbers in the address as well.

Using JavaScript Event Handlers

In order for your Web pages to interact with users, your scripts must be able to recognize and respond to user actions. Each action by a user is known as an **event**. JavaScript can respond to a user's actions with **event handlers**, which are terms that specify possible user actions. By including an event handler along with a set of instructions, your script can respond to certain user actions when they happen. Table H-2 lists 12 event handlers and describes the event that each one names. Lydia has read about using event handlers. She wants the Nomad Ltd home page to display a message in the status bar window when a user moves the pointer over the e-mail address. She recognizes that this pointer positioning is an event, and that she can use a script containing an event handler to display the message.

Steps 1 2 3 4

1. Open HTML H-2.htm in your text editor and save it as a text document with the filename **Event handler.htm**
   Lydia wants to insert her script code in the <A> code near the bottom of the page.
2. Scroll to the end of the document, select the text **[insert event handler here]** in the <A> tag, then press **[Delete]**
3. Type the following all on the same line, without pressing [Enter]:
   onMouseOver="window.status='We will reply to your inquiry within 24 hours!';return true" onMouseOut="window.status='';return true"
   The completed script is shown in Figure H-5. Even without pressing Enter, your text editor may wrap the text onto multiple lines. This will not affect the accuracy of your code. The window.status object refers to the status bar. In this script, the onMouseOver event handler instructs the browser to display the text typed between the apostrophes in the status bar when the pointer is over the hypertext link. In this script, the onMouseOut event handler instructs the browser to clear the text—that is, to display nothing in the status bar—when the mouse pointer is moved off the hypertext link. The text is cleared as indicated by the apostrophes containing no text or spaces that follow the second window.status.
4. Check the document for errors, make changes as necessary, then save Event handler.htm as a text document
5. Open the file **Event handler.htm** in your Web browser, then scroll to the bottom of the page
   Notice that the status bar contains information about the loading status of the current Web page.
6. Move the mouse pointer over the link **Tourdirector@nomadltd.com**, but do not click
   The scripted message appears in the status bar, as shown in Figure H-6.
7. Move the mouse pointer off the link
   The message no longer appears in the status bar.

Trouble?

Be sure you use two apostrophes, rather than a quotation mark, following *onMouseOut="window.status=.*

Trouble?

If your message does not appear correctly, check to be sure you have typed the script exactly as shown in Step 3.

FIGURE H-5: Completed script containing new event handlers

```
<H2>Tell Us More About Yourself!</H2>
<FORM NAME="info">
First Name:<INPUT TYPE="TEXT" SIZE=20 NAME="firstname">
Last Name: <INPUT TYPE="TEXT" SIZE=20 NAME="lastname"><BR>
Street Address:<INPUT TYPE="TEXT" SIZE=50 NAME="address"><BR>
City: <INPUT TYPE="TEXT" SIZE=20 NAME="city">
State:<INPUT TYPE="TEXT" SIZE=6 NAME="state">
Zip Code:<INPUT TYPE="TEXT" SIZE=15 NAME="zipcode">
<BR>Click here to submit this information. <INPUT TYPE="BUTTON" VALUE="Send
now!" onClick="submitted()">
</FORM>

<H2>Contact Us Directly!</H2>

Find out more about what we have to offer. Contact the tour
office at <A HREF="MAILTO:Tourdirector@nomadltd.com"
onMouseOver="window.status='We will reply to your inquiry within 24
hours!';return true" onMouseOut="window.status='';return
true">Tourdirector@nomadltd.com</A>

</BODY>
</HTML>
```

Event handler code entered

FIGURE H-6: Browser displaying status bar message

Event handler activated by position of pointer

Message in status bar triggered by event handler

TABLE H-2: JavaScript event handlers

event handler	triggering action	event handler	triggering action
onAbort	Page loading halted	**onLoad**	Page or image opens
onBlur	Object not current or highlighted	**onMouseOut**	Mouse pointer not over link
onChange	Object value changed	**onMouseOver**	Mouse pointer over link
onClick	Hyperlink or button clicked	**onSelect**	Text selected
onError	Error executing script	**onSubmit**	Form submitted
onFocus	Object current or highlighted	**onUnload**	Different page opened

Creating a Function

As your scripts become more complex, you will begin to incorporate many lines of code in each one to store and process information. To help keep your scripts organized, JavaScript allows you to group and name sets of script code in units called **functions**. A function is a set of code that performs a specific task. When you group the lines of your scripts into functions, the code is logically broken down into functional units, which makes it easier to understand and to debug. You usually define functions in a page's head section, which allows any programmer to quickly scan a page's code. Additionally, because each function has a name, you can easily refer to it in several different parts of a Web page. This means you do not need to duplicate code each time you want your page to repeat a procedure. Lydia wants to add a button to the Web page that will clear the user's input in the form. She writes a function to perform this task.

1. Open the file **HTML H-3.htm** in your text editor, then save it as a text document with the filename **Function.htm**
   Generally, it's best to define functions at the beginning of a Web page, to make sure they are available for all subsequent uses.

2. Select the text **[replace with clearUp function]** in the page's head section, then press **[Delete]**

3. Type the following, pressing **[Enter]** at the end of each line

```
function clearUp() {
    document.info.elements[0].value=""
    document.info.elements[1].value=""
    document.info.elements[2].value=""
    document.info.elements[3].value=""
    document.info.elements[4].value=""
    document.info.elements[5].value=""
```

   These lines define a function that assigns the value **null**, or nothing, to each of the six text fields on the page. This null value is defined with the paired quotation marks that follow each equal sign and that contain nothing between them.

4. Type **}**
   Figure H-7 shows the completed function, including the opening and closing braces. The code for a function is always demarcated with braces {}.

5. Scroll to the end of the document, select the text **[replace with Clear button code]**, then press **[Delete]**

6. Type **<INPUT TYPE="BUTTON" value="Clear form" onClick = "clearUp()">**
   The button tag includes an event handler to trigger, or **call**, the function clearUp() when a user clicks the button. Figure H-8 shows the completed button tag in the document.

7. Check the document for errors, make changes as necessary, then save Function.htm as a text document

8. Open **Function.htm** in your Web browser, fill in the six input fields, then click the **Clear form button**
   The browser clears each of the text input fields.

FIGURE H-7: Document containing clearUp() function code

```
<HTML>

<HEAD>
<TITLE>Nomad Ltd online</TITLE>

<SCRIPT LANGUAGE="javascript">
<!--
function submitted() {
      alert("Information submitted!")
}

function clearUp() {
      document.info.elements[0].value=""
      document.info.elements[1].value=""
      document.info.elements[2].value=""
      document.info.elements[3].value=""
      document.info.elements[4].value=""
      document.info.elements[5].value=""
}


//-->
</SCRIPT>

</HEAD>
```

Code defining new function

FIGURE H-8: Document containing clearUp() function reference

```
</UL>

<H2>Tell Us More About Yourself!</H2>
<FORM NAME="info">
First Name:<INPUT TYPE="TEXT" SIZE=20 ID="firstname">
Last Name: <INPUT TYPE="TEXT" SIZE=20 ID="lastname"><BR>
Street Address:<INPUT TYPE="TEXT" SIZE=50 ID="address"><BR>
City: <INPUT TYPE="TEXT" SIZE=20 ID="city">
State:<INPUT TYPE="TEXT" SIZE=6 ID="state">
Zip Code:<INPUT TYPE="TEXT" SIZE=15 ID="zipcode">
<BR>Click here to submit this information. <INPUT TYPE="BUTTON" VALUE="Send
now!" onClick="submitted()">
<BR>Click here to clear the form and start over.
<INPUT TYPE="BUTTON" VALUE="Clear form" onClick="clearUp()">
</FORM>

<H2>Contact Us Directly!</H2>

Find out more about what we have to offer. Contact the tour
office at <A HREF="MAILTO:Tourdirector@nomadltd.com"
onMouseOver="window.status='We will reply to your inquiry within 24
hours!';return true" onMouseOut="window.status='';return
true">Tourdirector@nomadltd.com</A>

</BODY>
</HTML>
```

Code referencing new function

Assigning a Variable

In scripting, you often instruct JavaScript to perform functions on pieces of information that you specify, known as **values**. For example, the text *We will reply to your inquiry within 24 hours!*, which you used earlier in conjunction with an event handler, is a value. Values can also include "true" and "false" and information about the user's browser, such as the width of the window in pixels. JavaScript can add or manipulate any numeric value mathematically. When values are composed of many characters, such as the message text above, or when they change in different situations, such as browser window dimensions, they can be cumbersome if you need to enter them several times in your script. You can make this process easier and more efficient by assigning the value to a **variable**, which serves as a nickname. When you assign a value to a variable, you enter or look up the value only one time and then you use the variable to refer to the value. Using variables saves you time when writing scripts. Variables provide added flexibility by allowing you to modify the value in only one place and have the modifications reflected instantaneously throughout the document as indicated by the variable. Lydia has modified the Nomad Ltd home page to personalize text by displaying the user's name. To finish her changes, she needs to define the variable she will use to represent the user's name, and then insert the variable name in the scripts.

Steps

1. Open the file **HTML H-4.htm** in your text editor, then save it as a text document with the filename **Variable.htm**
2. Select the text **[replace with variable definition]** near the top of the page, then press **[Delete]**
3. Type the following text on one line, without pressing [Enter]
   var name=prompt("For personal service, please type your first name and click OK.","")
   This line of script creates a dialog box that prompts the user to enter his or her first name and assigns the user's input to a variable named "name". Figure H-9 shows the document with this line of code inserted. The command "var" tells JavaScript that you are specifying a variable. The word following var—in this case, "name"—is the name of the variable you are creating. The value of the new variable follows the name after an equals sign. In this case, the value will be the result of user input in a dialog box created by the "prompt" method.
4. Scroll down to the beginning of the page's body section, select the text **[replace with heading script]**, then press **[Delete]**
   Notice that Lydia has already inserted the opening and closing script tags.

Trouble?

Be sure you type a space after the text **page**, and before the closing ".

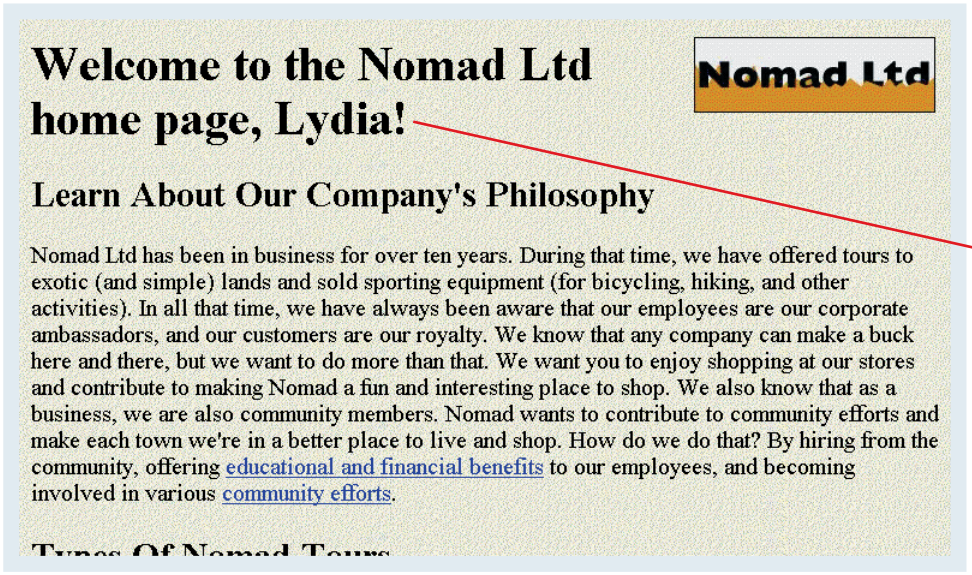
5. Type the following text on one line, without pressing [Enter]
   document.write("<H1>Welcome to the Nomad Ltd home page, " + name + "!</H1>")
   This script writes a line of code to the Web page, containing H1 tags to format the text. Lydia uses the variable "name" to insert the user's name into the page heading.
6. Scroll down to the beginning of the form section, select the text **[replace with second heading script]**, then press **[Delete]**
7. Type the following all on one line, without pressing [Enter]
   document.write("<H2>Please tell us more about yourself, " + name + ".</H2>")
   This script places the user's name at a second location in the Web page.
8. Check the document for errors, make changes as necessary, then save Variable.htm as a text document
9. Open the file **Variable.htm** in your Web browser, type your first name in the prompt dialog box, then click **OK**
   The Nomad Ltd home page is now personalized. The name the user enters in the prompt dialog box is displayed. See Figure H-10.

FIGURE H-9: Script creating a variable

```
<HTML>

<HEAD>
<TITLE>Nomad Ltd online</TITLE>

<SCRIPT LANGUAGE="javascript">
<!--
var name=prompt("For personal service, please type your first name and click
OK.","")

function submitted() {
      alert("Information submitted!")
}

function clearUp() {
      document.info.elements[0].value=""
      document.info.elements[1].value=""
      document.info.elements[2].value=""
      document.info.elements[3].value=""
      document.info.elements[4].value=""
      document.info.elements[5].value=""
}

//-->
</SCRIPT>
```

Variable inserted in script

FIGURE H-10: Web page incorporating user information



User name incorporated into Web page text generated by script

Manipulating variables

In addition to simply using variable values that a user enters or that a script looks up, your scripts can process values using **arithmetic operators**, which allow you to manipulate variables mathematically to create new values. For example, to count page headings, you could create a script that reads through your Web page code, adding 1 to a variable value each time it encounters a heading tag. You also could combine several values that a user enters using a mathematical equation.

Creating a Conditional

Sometimes, you want a script to be able to create different results depending on different user actions or on the value of a certain browser attribute. JavaScript allows you to set up this situation by creating a **conditional** in your script. A conditional allows your script to choose one of two paths, depending on a condition that you specify. For example, you might want a graphic to display at a smaller size if a user's window is not maximized, to keep it fully in view. You could use a conditional to check the dimensions of the user's browser window and then set the graphic dimensions to one of two preset choices. Conditionals allow you to create flexible, interactive scripts whose output can change in different situations. Nomad Ltd has had a problem with users submitting forms missing the zip code information. Lydia wants her Web page to verify that the user has completed the zip code form field before it submits the data to the server. Using a conditional, her page can prompt the user to complete the field if it is left blank.

Steps

1. Open the file **HTML H-5.htm** in your text editor, then save it as a text document with the filename **Conditional.htm**
2. Select **[replace with verify function]** in the page's head section and press **[Delete]**
3. Type the following code, pressing **[Enter]** at the end of each line

```
function verify() {
      if (document.info.elements[5].value == "") {
            alert("Please complete each field.")
      }     else {
            submitted()
      }
}
```

   Figure H-11 shows the document containing the verify function. The "if" statement code checks if the value of the zipcode field is null, indicating that the user has left it blank. When the "if" statement returns a value of "true," the function executes the code that immediately follows it. Here, the "true" result triggers the "alert" command to create a dialog box prompting the user to complete each field. When the "if" statement returns a value of "false," the function executes the code following the word *else*. In Lydia's script, the else command runs a function called "submitted()" that sends the user's information to the Web server.
4. Scroll to the bottom of the form section, select the text **[replace with event handler]** in the tag for the Send now! button, then press **[Delete]**
5. Type **onClick="verify()"**
6. Check scripts you added for errors, make changes as necessary, then save **Conditional.htm** as a text document
7. Open **Conditional.htm** in your Web browser, enter your name, scroll down to the form, then click the **Send now! button** without filling in any of the fields
   The script runs and displays the dialog box shown in Figure H-12.
8. Click the **OK button**, then fill in each of the fields with sample information and click the **Send now! button**
   The script you added verifies that the zip code field contains information and then allows the form submit function to execute.

FIGURE H-11: Web document containing completed script

Condition

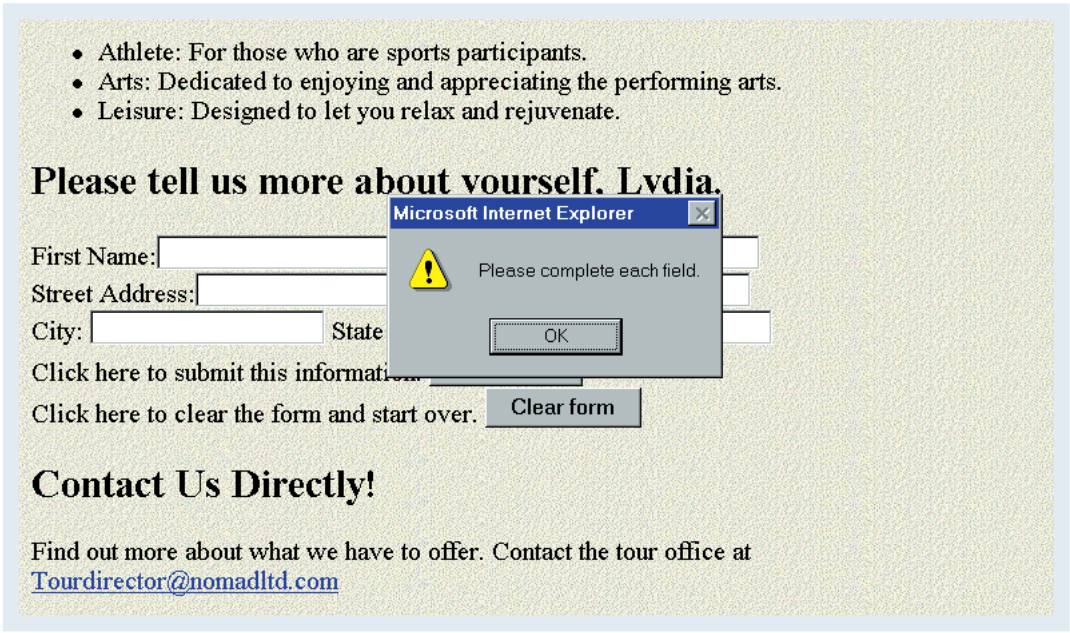
Conditional terms

"false" result

"true" result

```
        document.info.elements[5].value=""
}

function verify() {
        if (document.info.elements[5].value == "") {
                alert("Please complete each field.")
        }       else {
                submitted()
        }
}

//-->
</SCRIPT>

</HEAD>

<BODY BACKGROUND="egg shell.jpg">

<IMG SRC="nomad.jpg" ALIGN="right" WIDTH=201 HEIGHT=63>

<SCRIPT LANGUAGE="javascript">
<!--
document.write("<H1>Welcome to the Nomad Ltd home page, " + name + "!</H1>")
//-->
</SCRIPT>
```

FIGURE H-12: Browser showing alert dialog box



CLUES TO USE

Testing multiple conditions

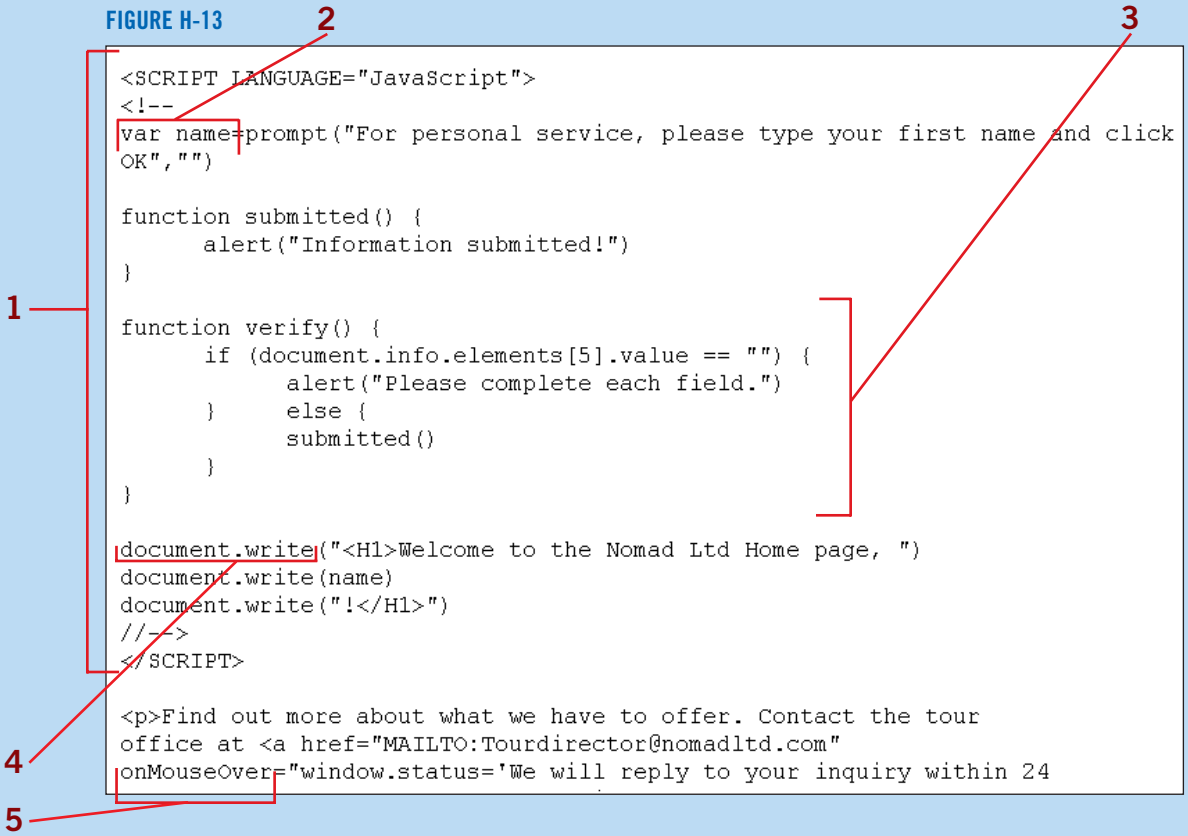
In addition to using a conditional to test a single condition, you test multiple conditions using logical comparison operators. JavaScript recognizes three logical comparison operators: && ("and"), || ("or"), and ! ("not"). A conditional using the && operator between two or more conditions returns "true" only if all conditions are true. Linking multiple conditions with the || operator, however, returns "true" if any one of the conditions is true. The ! operator returns true if its associated condition is not true.

Practice

Concepts Review

Label each item shown in Figure H-13.

FIGURE H-13



Match each statement with the term it describes.

6. Object hierarchy
7. Event handler
8. Bug
9. Method
10. JavaScript

a. Term specifying a possible user action
b. An error in a script
c. A Web page scripting language
d. JavaScript's object organization
e. Any action an object can carry out

Select the best answer from the list of choices.

11. What HTML tagset marks the beginning and end of a script?
- **a.** <JS>..</JS>
- **b.** <JAVASCRIPT>..</JAVASCRIPT>
- **c.** <SCRIPT>..</SCRIPT>
- **d.** <JAVASCRIPT>..<JAVASCRIPT>

12. Which scripting language is compatible with both Netscape Navigator 4 and Microsoft Internet Explorer 4?
- **a.** JScript
- **b.** JavaScript
- **c.** HTML
- **d.** VBScript

13. Which of the following typing mistakes would not result in an error in your script?
- **a.** Substituting the letter l for the number 1
- **b.** Inserting an extra space
- **c.** Omitting a closing bracket
- **d.** All of the above could result in errors.

14. What would be the proper form of address in the object hierarchy for the second element in a form called "info"?
- **a.** document.info.elements[1]
- **b.** document.info.elements[2]
- **c.** document.forms.info.elements[2]
- **d.** info.elements[2]

15. What is the function of the string <!-- in scripting?
- **a.** It is the HTML code for the beginning of a script.
- **b.** It is the HTML code for the end of a script.
- **c.** It tells an incompatible browser to ignore the code that follows.
- **d.** It tells the browser to display the text "--"

16. Which comparison operator returns true only if the values before and after are both true?
- **a.** &&
- **b.** ||
- **c.** !
- **d.** ==

17. A set of script code grouped logically and named is called a(n)
- **a.** Object.
- **b.** Variable.
- **c.** Hierarchy.
- **d.** Function.

Skills Review

1. **Create a script.**
   a. Open the text editor, open the file HTML H-6.htm, then save it as a text document with the filename Script review.htm.
   b. Select the text [replace with script tags] in the page's head section, then press [Delete].
   c. Type <SCRIPT LANGUAGE="javascript"> and press [Enter].
   d. Type <!-- and press [Enter] twice.
   e. Type //--> and press [Enter], then type </SCRIPT>.
   f. Save Script review.htm as a text document.
   g. Open Script review.htm in your Web browser.

2. **Debug a script.**
   a. Open the file HTML H-7.htm in your browser, read the description of the error in the JavaScript Error dialog box, then click yes.
   b. Open the file HTML H-7.htm in your text editor, then save it as a text document with the filename Debug review.htm.
   c. Position the pointer in the third line of the clearUp() function, which begins document.info.elements[2].value, then insert a second quotation mark (") after the existing one at the end of the line.
   d. Save Debug review.htm as a text document, open Debug review.htm in your Web browser, enter information in the form, then click the Clear form button.

3. **Use JavaScript event handlers.**
   a. Open the file HTML H-8.htm in your text editor, then save it as a text document with the filename Event handler review.htm.
   b. Scroll to the bottom of the document, select the text "[replace with event handler]" in the first <A> tag, then press [Delete].
   c. Type the following without pressing [Enter]:
   onMouseOver="window.status='Guidelines on scheduling vacation time';return true"
   onMouseOut="window.status='';return true"
   d. Review your typing for errors, then save Event handler review.htm as a text document.
   e. Open Event handler review.htm in your Web browser, move the mouse pointer over the hypertext link "Additional vacation information" and verify that the message "Guidelines on scheduling vacation time" appears in the status bar.
   f. If necessary, fix any errors in your text editor, then save and preview the page.

4. **Create a function.**
   a. Open the file HTML H-9.htm in your text editor, then save it as a text document with the filename Function review.htm.
   b. Select the text "[replace with date function]" in the page's head section, then press [Delete].
   c. Type function writeDate() {
   d. Press [Enter], then type document.write(month + "/" + today.getDate() + "/" + today.getYear() + ".")
   e. Press [Enter], then type }

f. Select the text [replace with function call] near the bottom of the page, then type writeDate()
g. Save Function review.htm as a text document.
h. Open Function review.htm in your browser, verify that it displays the current date below the list of vacation dates, use your text editor to make changes if necessary, then save Function review.htm as a text document.

5. Assign a variable.

a. Open the file HTML H-10.htm in your text editor, then save it as a text document with the filename Variable review.htm.
b. Select the text [replace with variable code] in the page's head section, then type var name=prompt("Please type your first name and click OK","").
c. Select the text [replace with variable reference] at the top of the body section, then press [Delete].
d. Type document.write(name).
e. Save Variable review.htm as a text document, then open Variable review.htm in your browser.
f. Type your first name, then click OK.
g. Use your text editor to debug as needed.

6. Create a conditional.

a. Open the file HTML H-11.htm in your text editor, then save it as a text document with the filename Conditional review.htm.
b. Select the text [replace with conditional function] in the page's head section, then press [Delete].
c. Type the following code, pressing [Enter] at the end of each line:

```
function verify() {
        if (document.info.elements[0].value == "" ||
        document.info.elements[1].value == "" ||
        document.info.elements[2].value == "" ||
        document.info.elements[3].value == "" ||
        document.info.elements[4].value == "" ||
        document.info.elements[5].value == "") {
                alert("Please complete each field.")
        }       else {
                submitted()
        }
}
```

d. Review your code for typing errors, then save Conditional review.htm as a text document.
e. Open Conditional review.htm in your browser, fill in every field in the form except the last name field, click the "Submit now!" button, verify that your browser opens a dialog box asking you to complete all the fields, then click OK.
f. Close your text editor and browser.

Independent Challenges

1. Green House, a local plant store, has hired you to add interactive features to their Web pages. You've decided to start by using an event handler to display link explanations in the status bar when users point to the page's links.

To complete this independent challenge:

a. Start your text editor program, open the file HTML H-12.htm, then save it as a text document with the filename Green House home.htm.

b. To define the text strings that will be displayed as variables, select the text [replace with variable script] at the top of the page, press [Delete], and insert the following script, pressing [Enter] at the end of each line:

```
<SCRIPT LANGUAGE="javascript">
<!--
var plants="An overview of our plant stock and sources"
var tips="Helpful growing hints on common houseplants"
var services="A guide to our professional plant care services"
//-->
</SCRIPT>
```

c. Select the text "[replace with first event handlers]" in the <A> tag for the first link at the bottom of the page, press [Delete], and type the following without pressing [Enter]:

```
onMouseOver="window.status=plants;return true"onMouseOut="window.status='';return true"
```

d. Repeat Step c to modify the second link with the following insertion:

```
onMouseOver="window.status=tips;return true" onMouseOut="window.status='';return true"
```

e. Repeat Step c to modify the third link with the following insertion:

```
onMouseOver="window.status=services;return true" onMouseOut="window.status='';return true"
```

f. Save Green House home.htm as a text document, then open the file Green House home.htm in your Web browser.

g. Use your text editor to make any changes necessary, always save the file as a text document, then close your text editor and browser.

2. You have designed a Web publication for Sandhills Regional Public Transit. You want to incorporate a function you've written to tell prospective riders which fare period is in effect. A sentence stating the fare period will display at the bottom of the Web page. This sentence will change depending on the time of day.

To complete this independent challenge:

a. Start your text editor program, open the file HTML H-13.htm, then save it as a text document with the filename SRPT home.htm.

b. Replace the text [replace with fare function] in the page's head section with the script located in the file HTML H-14.txt.

c. Scroll down immediately before the closing body section tag, delete the text [replace with function call script], enter the two opening script tags, type "schedTime()" (without the quotes) as the third line of the script, then enter the two closing script tags.

d. Save and preview your file in your Web browser program, debugging as necessary until you see no more JavaScript error messages.

3. You have been hired by Community Public School Volunteers to add advanced features to their Web publication using scripts. You have inserted a script into the home page they provided, but a JavaScript error dialog box opens when the page loads. To complete this independent challenge, preview the file HTML H-15.htm in your browser, noting the type of JavaScript error described and its location. Open the page in your text editor and save a copy as CPSV home.htm. Edit the script to fix the JavaScript error. Use the guidelines listed earlier in the unit to identify the types of errors that may be present. Continue to preview and edit the page until you no longer receive an error message, then save your work in your text editor as a text file.

4. Scripts are used in many Web pages on the WWW today to add features, as well as to create interesting formatting that is not possible with standard HTML coding. To complete this independent challenge, find two pages on the Web that contain scripts. (*Hint:* after opening a page in your browser, choose the HTML or Page Source option in your browser's View menu to check the document code for <SCRIPT> tags.) Print their HTML source code, and circle all scripts you see on your printouts. Circle and label any variables, conditionals, and functions you see in the documents.

Visual Workshop

Open your text editor program, open the file HTML H-16.htm, and save it as a text document named Touchstone.htm. Then open Touchstone.htm in your browser. Scroll down to see the Send now! button. Click the Send now! button on the form without entering information in any of the fields. Use the form-verification script located in the file HTML H-17.txt and your text editor to modify the document Touchstone.htm. Change the event handler for the Send now! button to run the verify() function. Save the changes as a text document with the filename Touchstone.htm. Click the browser program button on the taskbar, and refresh your screen. Click the Send now! button again. Your screen should look like Figure H-14. Debug as necessary. (*Hint:* Remember to insert the verification script in the document header section between the beginning and ending script tags.)

FIGURE H-14